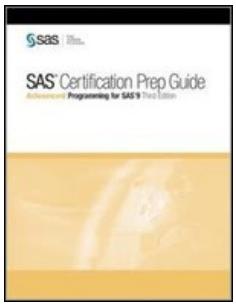



# SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute



---

# Chapter 6: Creating and Managing Indexes Using PROC SQL

## Overview

### Introduction

When processing a query that contains a subsetting WHERE clause or that joins multiple tables, PROC SQL must locate specific rows in the referenced table(s). Creating an index for a table enables PROC SQL, in certain circumstances, to locate specific rows more quickly and efficiently. An *index* is an auxiliary file that stores the physical location of values for one or more specified columns *(key columns)* in a table. In an index, each unique value of the key column(s) is paired with a location identifier for the row that contains that value. In the same way that you use a book's subject index to find a page that discusses a particular subject, PROC SQL uses the system of directions in an index to access specific rows in the table directly, by index value. You can create more than one index for a single table. All indexes for a SAS table are stored in one index file.

**Note** You cannot create an index on a view.

The following PROC SQL step uses the CREATE INDEX statement to create an index for a table, and uses the DESCRIBE TABLE statement to display information about the index, along with other information about the table, in the SAS log:

```
proc sql;
   create unique index empid
      on work.payrollmaster(empid);
   describe table work.payrollmaster;
```

**Note** For more information about the BUFSIZE= option, see "Using the BUFSIZE = Option" on page 715.

**Table 6.1: SASLog**

```
create table WORK.PAYROLLMASTER(bufsize=4096)
  (
   DateOfBirth num format=DATE9. informat=DATE9.,
   DateOfHire num format=DATE9. informat=DATE9.,
   EmpID char(4),
   Gender char(1),
   JobCode char(3),
   Salary num format=DOLLAR9.
  );
create unique index EmpID on WORK.PAYROLLMASTER(EmpID);
```

In this chapter, you will learn to use PROC SQL to create and manage various types of indexes.

### Objectives

In this chapter, you learn to

- determine when it is appropriate to create an index
- create simple and composite indexes on a table
- create an index that ensures that values of the key column(s) are unique
- control whether PROC SQL uses an index or which index it uses
- display information about the structure of an index in the SAS log
- drop (delete) an index from a table.

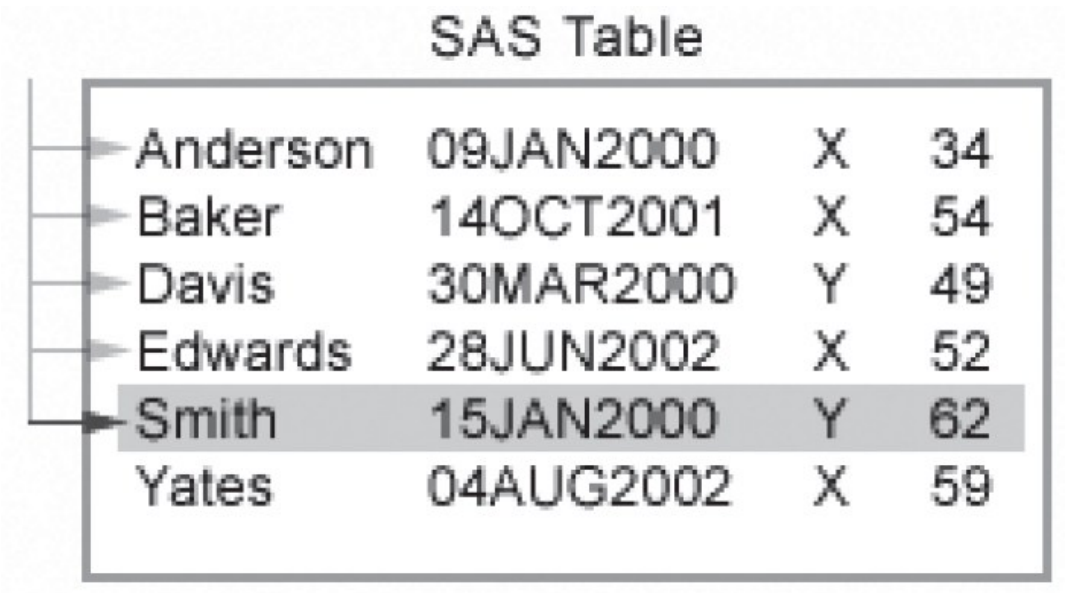
### Prerequisites

Before reading this chapter, you should complete the following chapters:

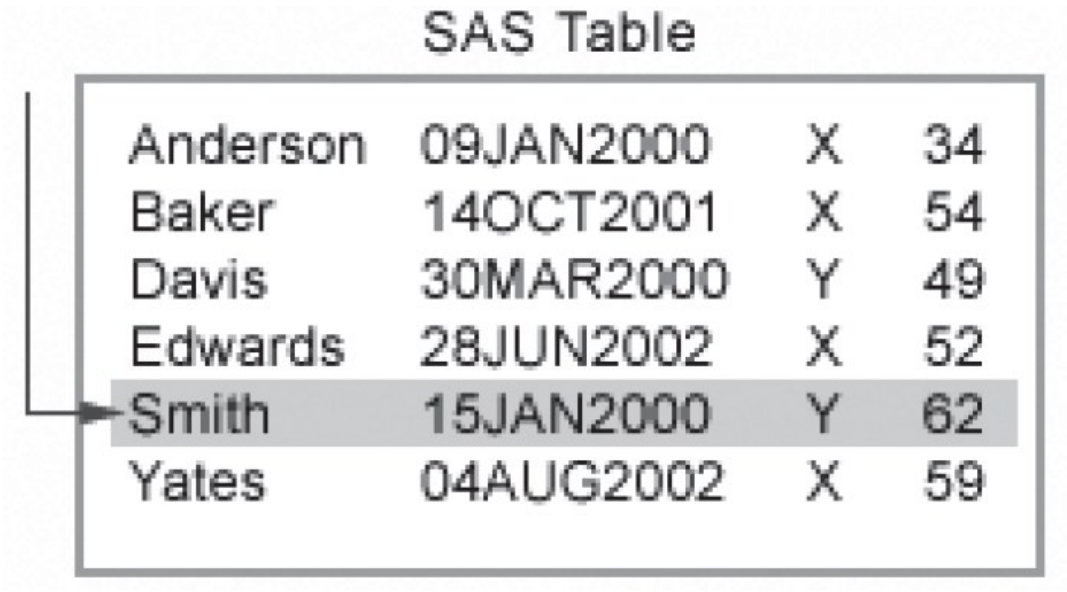
## Understanding Indexes

### Accessing Rows in a Table

When you submit a query on a table that does *not* have an index, PROC SQL *accesses rows sequentially*, in the order in which they are stored in the table. For example, suppose you are working with a table that contains information about employees. You have written a PROC SQL query to select the rows in which the value of **`Name`** (the first column) is *Smith*. To access the rows that you want, PROC SQL begins with the first row and reads through all rows in the table, selecting the rows that satisfy the condition that is expressed in the WHERE clause.




When you execute a program that retrieves a small subset of rows from a large table, it can be time-consuming for PROC SQL to read the rows sequentially. In some situations, using an index on a table allows PROC SQL to access a subset of rows more efficiently. An index stores unique values for a *specified column or columns* in ascending value order, and includes information about the location of those values in the table. That is, an index includes *value/identifier pairs* that enable you to *access a row directly*, by value. For example, suppose you have created an index on your table that is based on the column **`Name.`** Using the index, PROC SQL will access the row(s) that you want directly, without having to read all the other rows.




### Simple and Composite Indexes

You can create two types of indexes:

- simple
- composite.

A *simple index* is based on one column that you specify. The indexed column can be either character or numeric. When you create a simple index by using PROC SQL, you must specify the name of the indexed column as the name of the index.

A *composite index* is based on two or more columns that you specify. The indexed columns can be character, numeric, or a combination of both. In the index, the values of the key columns are concatenated to form a single value.

For example, if you build a composite index on the key columns **`LastName`** and **`FirstName`**, a value for the index is composed of the value for **`LastName`** followed by the value for **`FirstName.`** Often, a WHERE clause might use only the first column (the primary key) of a composite index, which means that the program will read only the first part of each concatenated value.

When you create a composite index, you must specify a unique name for the index that is not the name of any existing column or index in the table. In the example described above, the composite index *cannot* be named *Lastname* or *Firstname*.

### Unique Indexes

If you want to require that values for the key column(s) are unique for each row, you can create either a simple or a composite index as a *unique index*. Once a unique index is defined on one or more columns in a table, SAS will reject any change to the table that would cause more than one row to have the same value(s) for the specified column or composite group of columns.

### Example

Suppose you are working with the table *Sasuser.payrollmaster*. The first eight rows of this table are shown below.

| DateOfBirth | DateOfHire | EmpID | Gender | JobCode | Salary |
|---|---|---|---|---|---|
| 16SEP1958 | 07JUN1985 | 1919 | M | TA2 | $48,126 |
| 190CT1962 | 12AUG1988 | 1653 | F | ME2 | $49,151 |
| 08NOV1965 | 190CT1988 | 1400 | M | ME1 | $41,677 |

| | | | | | |
|---|---|---|---|---|---|
| 04SEP1963 | 01AUG1988 | 1350 | F | FA3 | $46,040 |
| 19DEC1948 | 21NOV1983 | 1401 | M | TA3 | $54,351 |
| 29APR1952 | 11JUN1978 | 1499 | M | ME3 | $60,235 |
| 09JUN1960 | 040CT1988 | 1101 | M | SCP | $26,212 |
| 03APR1959 | 14FEB1979 | 1333 | M | PT2 | $124,048 |

If you know that the column **JobCode** is often specified in a WHERE clause expression, you might want to create a *simple index* on the column **JobCode.** You must specify the name of the key column, **JobCode**, as the index name.

Now suppose you are planning to write many queries that specify both **EmpID** and **DateOfHire** in a WHERE clause expression. In this case, you might want to create a *composite index* on these two columns. Because employee identification numbers should be unique, it is appropriate to create this index as a unique index. Therefore, you should specify a name for your index that is not the same as the name of any existing column or index in the table. For example, you could name this index *Whenhired*.

## Deciding Whether to Create an Index

### Overview

An index can reduce the time required to locate a set of rows, especially for a large data file. However, there are costs associated with creating, storing, and maintaining the index. When deciding whether to create an index, you must weigh any benefits in performance improvement against the costs of increased resource usage.

**Note** This chapter discusses the benefits and costs that are associated with using indexes specifically with PROC SQL. To learn about the costs and benefits of using indexes with other SAS procedures, see the SAS documentation.

### PROC SQL Queries That Can Be Optimized by an Index

To use indexes effectively with PROC SQL, it is important to know the classes of queries that can be processed more efficiently by using an index. The classes of queries that can be optimized are specified below.

| Query performance is optimized when the key column occurs in … | Example |
|---|---|
| a *WHERE clause expression* that contains<br>▪ a comparison operator<br>▪ the *TRIM* or *SUBSTR function*<br>▪ the CONTAINS operator<br>▪ the *LIKE operator*. | `proc sql;`<br>`  select empid, jobcode, salary`<br>`  from sasuser.payrollmaster`<br>`  where jobcode='FA3'`<br>`  order by empid;`<br>*Key Column(s):* **JobCode** |
| a subquery returning values to the *IN operator*. | `proc sql;`<br>`  select empid, lastname, firstname,`<br>`  city, state`<br>`  from sasuser.staffmaster`<br>`  where empid in`<br>`  (select empid`<br>`  from sasuser.payrollmaster`<br>`  where salary>40000);`<br>*Key Column(s):* **EmpID** |
| a *correlated subquery*, in which the column being compared with the correlated reference is indexed | `proc sql;`<br>`  select lastname, firstname`<br>`  from sasuser.staffmaster`<br>`  where 'NA'=`<br>`  select jobcategory`<br>`  from sasuser.supervisors`<br>`  where staffmaster.empid =`<br>`  supervisors.empid);`<br>*Key Column(s):* **Supervisors.EmpID** |

- a *join* in which
- the join expression contains the *equals (=) operator* (an *equijoin)*
- all the columns in the join expression are indexed in one of the tables being joined.

```
proc sql;
    select *
       from sasuser.payrollmaster as p,
            sasuser.staffmaster as s
       where p.empid =
             s.empid
       order by jobcode;
```

*Key Column(s):* **`Payrollmaster.EmpID`** or **`Staffmaster.EmpID`**

### Benefits of Using an Index

For PROC SQL, there are three main benefits to using an index to access data directly (instead of reading the data sequentially):

- A *small subset of data* (<15% of rows) can be accessed more quickly. (As the size of the subset increases, the advantage of using an index decreases.)
- *Equijoins* can be performed without internal sorts.
- *Uniqueness* can be enforced by creating a unique index.

### Example: Using an Index to Access a Small Subset of Data

Suppose you are writing a query that references the table *Work.Payrollmaster. (Work.Payrollmaster* is a duplicate of the table *Sasuser.payrollmaster.) Work.Payrollmaster* stores payroll information for employees, and it has a simple index defined on the column **`JobCode.`** Your query's WHERE clause expression references the key column:

```
proc sql;
   select empid, jobcode, salary
      from work.payrollmaster
      where jobcode='FA3'
      order by empid;
```

If the value of **`JobCode`** for *most* of the rows in the table is *FA3*, then the use of an index will *not* significantly improve the efficiency of the following query. In fact, performance might be degraded.

However if only 10% of the rows have a value of *FA3*, then PROC SQL can process the query more efficiently by using the index.

**Note** In this chapter, if you want to submit any sample code that references a temporary table (a table that is stored in the *Work* library), you will first need to create the temporary table by copying the table in the *Sasuser* library that has the same name.

### Understanding the Costs of Using an Index

When you are deciding whether to create an index, you should consider the associated increase in resource usage, which includes the following:

- *Additional CPU time* is necessary to create an index, to maintain the index when the table is modified, and to use an index to read a row from a table.
- Using an index to read rows from a table might require *additional I/O (input/output) requests* when compared to reading the table sequentially.
- Using an index requires *additional memory* for buffers into which the index pages and code are loaded for processing.
- *Additional disk space* is required to store the index file, which can show up as a separate file (in the Windows and UNIX operating environments, for example) or can appear to be part of the data file (in the z/OS operating environment).

### Guidelines for Creating Indexes

To use indexes effectively, follow these guidelines for creating indexes:

- Keep the number of indexes to a minimum to reduce disk storage and update costs.
- Do not create an index for small tables. Sequential access is faster on small tables.
- Do not create an index based on columns that have a very small number of distinct values, low cardinality (for example, a **Gender** column that contains only the two values *Male* and *Female*).
- Use indexes for queries that retrieve a relatively small subset of rows — that is, less than 15%.
- Do not create more than one index that is based on the same column as the primary key.

**Tip** Many factors affect the processing of SAS programs. The most accurate way to find out whether to create an index for a particular table or column is to perform benchmarking tests.

## Creating an Index

### Overview

To create an index on one or more columns of a table, use the CREATE INDEX statement.

---

General form, CREATE INDEX statement:

```
CREATE <UNIQUE> INDEXindex-name
        ON table-name(column-name-l<, ...column-name-n);
```

where

UNIQUE
is a keyword that specifies that all values of the column(s) specified in the statement must be unique.

*index-name*
specifies the name of the index to be created. If you are creating an index on one column only, then *index-name* must be the same as *column-name-1*. If you are creating an index on more than one column, then *index-name* cannot be the same as the name of any existing column or index in the table.

*table-name*
specifies the name of the table on which the index will be created.

*column-name*
specifies a column to be indexed. Columns can be specified in any order; however, column order is important for data retrieval. The first-named column is the primary key, the second-named column is the secondary key, and so on.

---

**Tip** When creating a composite index, specify the columns in the same order as you would specify them in an ORDER BY clause.

**Tip** You can achieve improved index performance if you create the index on a presorted table.

SAS maintains indexes for all changes to the table, whether the changes originate from PROC SQL or some other source, as long as the entire table is not re-created. If you alter a column's definition or update its values, then SAS will update the indexes also. However, if a key column in a table is dropped (deleted), then the index on that column is also dropped.

### Creating Multiple Indexes

You cannot create multiple simple indexes that are based on the same column or multiple composite indexes that are based on the same set of columns. Although it is possible to create both a simple and a composite index on the same column, it is usually not advantageous to do this. If a simple index is defined on a column and that column is also the primary key in a composite index, PROC SQL will use the composite index in processing a query that references that

column.

You can create multiple indexes on the same table, but you must use a separate CREATE INDEX statement for each index that you want to create.

### Example: Creating a Simple Index

The following PROC SQL step uses the CREATE INDEX statement to create a simple, unique index that is based on the column **EmpID** in the temporary table *Work.Payrollmaster*. (*Work.Payrollmaster* is a duplicate of the table *Sasuser.Payrollmaster*.)

```
proc sql;
   create unique index EmpID
      on work.payrollmaster(empid);
```

The specified index name (*EmpID*) must be the same as the name of the key column.

When this step is submitted, the SAS log displays the following message.

**Table 6.2: SAS Log**

```
NOTE: Simple index EmpID has been defined.
```

### Example: Creating a Composite, Unique Index

The following PROC SQL step uses the CREATE INDEX statement to create the composite, unique index *daily* on the columns **FlightNumber** and **Date:**

```
proc sql;
   create unique index daily
      on work.marchflights(flightnumber,date);
```

When this step is submitted, the SAS log displays the following message.

**Table 6.3: SAS Log**

```
NOTE: Composite index daily has been defined.
```

**Note** The note in the SAS log displays the index name exactly as you specified it. In this example, the index name *daily* was specified in lowercase. In the previous example, the index name *EmpID* was specified in mixed case. However, the use of uppercase and lowercase for index names is not significant because SAS recognizes index names regardless of how they are formatted in code.

If the set of key columns **FlightNumber** and **Date** had duplicate values, the index would *not* be created. Instead, the SAS log would display a message like the following.

**Table 6.4: SAS Log**

```
ERROR: Duplicate values not allowed on index daily for file MARCHFLIGHTS.
```

## Displaying Index Specifications

### Overview

Sometimes you want to know whether an existing table has any indexes. To display a CREATE INDEX statement in the SAS log for each index that is defined for one or more specified tables, you can use the DESCRIBE TABLE statement. (The DESCRIBE TABLE statement also writes a CREATE TABLE statement to the SAS log for each specified table.)

---

General form, DESCRIBE TABLE statement:

**DESCRIBE TABLE** *table-name-l<,… table-name-n>;*

where

*table-name*

specifies the table to be described as one of the following:

- a one-level name
- a two-level *libref.table* name
- a physical pathname that is enclosed in single quotation marks.

---

If a specified table has no indexes defined, a CREATE INDEX statement will not appear.

### Example

Earlier in this chapter, the following code was used to create a unique composite index named *daily* on the columns **`FlightNumber`** and **`Date`** in the temporary table *Marchflights*.

```
proc sql;
   create unique index daily
      on work.marchflights(flightnumber,date);
```

The following DESCRIBE TABLE statement writes a CREATE INDEX statement to the SAS log (after the CREATE TABLE statement) for the table *Marchflights:*

```
proc sql;
   describe table marchflights;
```

**Table 6.5: SAS Log**

```
NOTE: SQL table WORK.MARCHFLIGHTS was created like:

create table WORK.MARCHFLIGHTS(bufsize=8192)
  (
   Date num format=DATE9. informat=DATE9.,
   DepartureTime num format=TIME5. informat=TIME5.,
   FlightNumber char(3),
   Origin char(3),
   Destination char(3),
   Distance num,
   Mail num,
   Freight num,
   Boarded num,
   Transferred num,
   NonRevenue num,
   Deplaned num,
   PassengerCapacity num
  );
create unique index daily on WORK.MARCHFLIGHTS(FlightNumber,Date);
```

If the table *Marchflights* had no index defined, no CREATE INDEX statement would appear in the SAS log.

### Alternatives to the DESCRIBE TABLE Statement

The DESCRIBE TABLE statement is only one of several methods that can be used to list information about indexes that are defined on a table. One alternative is to query the special table *Dictionary.Indexes*, which contains information about indexes that are defined for all tables that are known to the current SAS session. *(Dictionary.Indexes* is one of many read-only dictionary tables that are created at PROC SQL initialization. These tables contain information about SAS libraries, SAS macros, and external files that are in use or available in the current SAS session.)

You can also use other SAS procedures, such as PROC CONTENTS and PROC DATASETS, to generate a report that contains information about indexes.

**Note** To learn more about the use of dictionary tables, see "Managing Processing Using PROC SQL" on page 278. To

learn more about using PROC CONTENTS and PROC DATASETS, see "Creating Samples and Indexes" on page 470.

## Managing Index Usage

### Overview

To manage indexes effectively, it is important to know

- how SAS decides whether to use an index and which index to use
- how to determine whether SAS is using an index
- how to control whether SAS uses an index, or which index it uses.

### Understanding How SAS Decides Whether to Use an Index

By default, each time you submit a query (or other SAS program) that contains a WHERE expression, SAS decides whether to use an index or to read all the observations in the data file sequentially. To make this decision, SAS does the following:

1. Identifies an available index or indexes.
2. Estimates the number of rows that would be qualified. If multiple indexes are available, SAS selects the index that it estimates will return the smallest subset of rows.
3. Compares resource usage to decide whether it is more efficient to satisfy the WHERE expression by using the index or by reading all the observations sequentially.

Next, consider how you can find out whether SAS is using an index.

### Determining Whether SAS Is Using an Index

After you create an index, it is important to monitor whether the index is being used. If an index is *not* being used, the costs of maintaining the index might be greater than the benefits, and you should consider dropping (deleting) the index.

By default, when a PROC SQL query or any other program is submitted in SAS, only notes, warnings, and error messages are written to the SAS log. To display additional messages, such as information about indexes that have been defined and that have been used in processing the program, specify the SAS system option *MSGLEVEL=I*. You specify the MSGLEVEL= option in the OPTIONS statement, before the PROC SQL statement.

---

General form, MSGLEVEL= option:

```
OPTIONS MSGLEVEL=N| I;
```

where

N
: displays notes, warnings, and error messages only. This is the default.

I
: displays additional notes pertaining to index usage, merge processing, and sort utilities along with standard notes, warnings, and error messages.

---

Usually, the option MSGLEVEL= is set to I for debugging and testing, and set to N for production jobs.

### Example: Query That Uses an Index

Suppose you are writing a PROC SQL query that references the temporary table *March/lights*. Earlier in this chapter, a unique composite index named *daily* was created on the columns **`FlightNumber`** and **`Date`** in *March/lights*. The WHERE expression in your query specifies the key column **`FlightNumber.`** To determine whether PROC SQL uses the index *daily*

when your query is processed, you specify MSGLEVEL=I before the query:

```
options msglevel=i;
proc sql;
   select *
      from marchflights
      where flightnumber='182';
```

The message in the SAS log shows that the index was used in processing.

**Table 6.6: SAS Log**

```
INFO: Index daily selected for WHERE clause optimization.
```

### Example: Query That Does Not Use an Index

Suppose you submit a different query that also references the key column **`FlightNumber`**:

```
proc sql;
   select *
      from marchflights
      where flightnumber in ('182','202');
```

In this example, the SAS log shows that the query does *not* use the index.

**Table 6.7: SAS Log**

```
INFO: Index daily not used. Sorting into index order may help.
INFO: Index daily not used. Increasing bufno to 8 may help.
```

**Note** For more information about the BUFSIZE= option, see "Using the BUFSIZE= Option" on page 715.

**Note** SAS Version 8 displays informational messages that indicate when an index *is* used, but does *not* display messages that indicate when an index is *not* used.

**Tip** Because the OPTIONS statement is global, the settings remain in effect until you modify them or until you end your SAS session. Therefore, you do not need to specify MSGLEVEL=I in this second query or any subsequent queries until you want to change the setting or until your SAS session ends.

### Controlling Index Usage

In general, it is recommended that you allow SAS to decide whether to use an index, or which index to use, in processing a PROC SQL query (or other SAS program). However, in some situations, such as testing, you might find it useful to control the use of indexes by SAS.

To control index usage, use the *IDXWHERE*= and *IDXNAME*= SAS data set options to override the default settings. You can use either of these options, but you cannot use both options at the same time. As with other SAS data set options, you specify the IDXWHERE= or IDXNAME= option in parentheses after the table name in the FROM clause of a PROC SQL query.

### Using IDXWHERE= to Direct SAS to Use or Not to Use an Index

The IDXWHERE= option enables you to override the decision that SAS makes about whether to use an index.

---

General form, IDXWHERE= option:

**`IDXWHERE`**`=YES | NO;`

where

YES

tells SAS to choose the best index to optimize a WHERE expression, and to disregard the possibility that a

sequential search of the table might be more resource-efficient.

NO

tells SAS to ignore all indexes and satisfy the conditions of a WHERE expression with a sequential search of the table.

---

**Note** Use the IDXWHERE=NO option when you know an available index will not optimize WHERE clause processing.

### Example

In an earlier example, you used the option MSGLEVEL=I to verify that PROC SQL *does* use an index to process the following query:

```
options msglevel=i;
proc sql;
   select *
      from marchflights
      where flightnumber='182';
```

To force SAS to ignore the index and to process the rows of the table sequentially, specify IDXWHERE=NO in the query:

```
proc sql;
   select *
      from marchflights (idxwhere=no)
      where flightnumber='182';
```

A message in the SAS log indicates that SAS was forced to process the data sequentially.

**Table 6.8: SAS Log**

```
INFO: Data set option (IDXWHERE=NO) forced a sequential pass of the data
rather than use of an index for where-clause processing.
```

### Using IDXNAME= to Direct SAS to Use a Specified Index

The IDXNAME= option directs SAS to use an index that you specify, even if SAS would have selected *not* to use an index or to use a different index.

---

General form, IDXNAME= option:

**IDXNAME**=*index-name;*

where

*index-name*

specifies the name of the index that should be used for processing.

---

SAS uses the specified index if the following conditions are true:

- The specified index must exist.
- The specified index must be suitable by having at least its first or only column match a condition in the WHERE expression.

**Note** Use the IDXNAME= option when you know the better index so SAS does not have to do the evaluation.

### Example

In an earlier example, a composite index named *daily* was defined on the columns **FlightNumber** and **Date** in the temporary table *March/lights*. Suppose you create a second index, a simple index, on the column **Date** (the secondary key in the composite index) by using the following PROC SQL step:

```
proc sql;
   create index Date
      on work.marchflights(Date);
```

Next, you submit the following query:

```
proc sql;
   select *
      from marchflights
      where date='01MAR2000'd;
```

The WHERE clause in this query references the key column **Date.** By default, SAS decides whether to use an index and, if an index is used, which index to use. The SAS log indicates that, with both a simple and a composite index defined on **Date**, PROC SQL used the simple index *Date* to process the query.

**Table 6.9: SAS Log**

```
INFO: Index Date selected for WHERE clause optimization.
```

**Note** This example assumes that the option MSGLEVEL=I, which was specified in the previous example, is still in effect.

You decide that you want to force PROC SQL to use the index *daily* instead of **Date**, so you add IDXNAME= to your query:

```
proc sql;
   select *
      from marchflights (idxname=daily)
      where flightnumber='182';
```

After this query is submitted, a message in the SAS log indicates that PROC SQL used the index *daily:*

**Table 6.10: SAS Log**

```
INFO: Index daily selected for WHERE clause optimization.
```

## Dropping Indexes

### Overview

To drop (delete) one or more indexes, use the *DROP INDEXstatement*.

---

General form, DROP INDEX statement:

```
DROP INDEX index-name-1 <, …index-name-2>
      FROM table-name;
```

where

*index-name*

specifies an index that exists.

*table-name*

specifies a table that contains the specified index(es). The *table-name* can be one of the following:

- a one-level name
- a two-level libref.table name
- a physical pathname that is enclosed in single quotation marks.

---

### Example: Dropping a Composite Index

The following PROC SQL step uses the DROP INDEX statement to drop the composite, unique index *daily* from the

temporary table *March/lights*. (This index was created in an example earlier in this chapter.)

```
proc sql;
   drop index daily
      from work.marchflights;
```

When this step is submitted, the SAS log displays a message indicating that the index has been dropped.

**Table 6.11: SAS Log**

```
NOTE: Index daily has been dropped.
```

## Summary

## Contents

This section contains the following topics.



## Text Summary

### Understanding Indexes

An index is an auxiliary file that is defined on one or more of a table's columns, which are called key columns. The index stores the unique column values and a system of directions that enable access to rows in that table by index value. When an index is used to process a PROC SQL query, PROC SQL accesses directly (without having to read all the prior rows) instead of sequentially.

You can create two types of indexes:

- simple index (an index on one column)
- composite index (an index on two or more columns).

You can define either type of index as a unique index, which requires that values for the key column(s) be unique for each row.

### Deciding Whether to Create an Index

When deciding whether to create an index, you must weigh any benefits in performance improvement against the costs of increased resource usage. Certain classes of PROC SQL queries can be optimized by using an index. To optimize the performance of your PROC SQL queries, you can follow some basic guidelines for creating indexes.

### Creating an Index

To create an index on one or more columns of a table, use the CREATE INDEX statement. To specify a unique index, you add the keyword UNIQUE.

### Displaying Index Specifications

To display a CREATE INDEX statement in the SAS log for each index that is defined for one or more specified tables, use the DESCRIBE TABLE statement.

### Managing Index Usage

To manage indexes effectively, it is important to know how SAS decides whether to use an index and which index to use.

To find out whether an index is being used, specify the SAS option MSGLEVEL=I in an OPTIONS statement before the PROC SQL statement. This option enables SAS to write informational messages about index usage (and other additional

information) to the SAS log. The default setting MSGLEVEL=N displays notes, warnings, and error messages only.

To force SAS to use the best available index, to use a specific index, or not to use an index at all, include either the SAS data set option IDXWHERE= or IDXNAME= in your PROC SQL query.

#### Dropping Indexes

To drop (delete) one or more indexes, use the DROP INDEX statement.

### Syntax

```
OPTIONS MSGLEVEL=N|I;
PROC SQL;
     CREATE <UNIQUE7>INDEX index-name
         ON table-name (column-name-l<, …column-name-n>);
     DESCRIBE TABLE table-name-l<, … table-name-n>;
     SELECT column-l<, … column-n>
          FROM table-1 (IDXWHERE=Y | N)
          <WHEREexpression>;
     SELECT column-l<, … column-n>
         FROM table-1 (IDXNAME=index-name)
         <WHEREexpression>;
     DROP INDEX table-name-l<, … table-name-n>;
          FROM table-name;
QUIT;
```

### Sample Programs

#### Creating a Simple, Unique Index and a Composite Index

```
proc sql;
   create unique index EmpID
      on work.payrollmaster(empid);
   create index daily
      on work.marchflights(flightnumber,date);
quit;
```

#### Displaying Index Specifications

```
proc sql;
   describe table marchflights;
quit;
```

#### Determining WhetherSAS Is Using an Index

```
options msglevel=i;
proc sql;
   select *
      from marchflights
      where flightnumber='182';
quit;
```

#### Directing SAS to Ignore All Indexes

```
proc sql;
   select *
      from marchflights (idxwhere=no)
      where flightnumber='182';
quit;
```

#### Directing SAS to Use a Specified Index

```
proc sql;
   select *
      from marchflights (idxname=daily)
      where flightnumber='182';
quit;
```

#### Dropping an Index

```
proc sql;
   drop index daily
```

```
      from work.marchflights;
quit;
```

### Points to Remember

- An index cannot be created on a view.
- Keep the number of indexes to a minimum to reduce disk storage and update costs.
- Do not create an index for small tables; sequential access is faster on small tables.
- Do not create an index based on columns that have a very small number of distinct values, low cardinality (for example, a **`Gender`** column that contains only the two values *Male* and *Female)*.
- Use indexes for queries that retrieve a relatively small subset of rows — that is, less than 15%.
- Do not create more than one index that is based on the same column as the primary key.

### Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

**1.** Which of the following will create an index on the column **`EmpID`** for the table *Sasuser.Staffmaster?* ?

a. a.
```
proc sql;
 create simple index(empid)
    on sasuser.staffmaster;
```

b. b.
```
proc sql;
 create empid index
    on sasuser.staffmaster(empid);
```

c. c.
```
proc sql;
 create simple index
    on empid from sasuser.staffmaster;
```

d. d.
```
proc sql;
 create index empid
    on sasuser.staffmaster(empid);
```

**2.** Which keyword must you add to your index definition in the CREATE INDEX statement to ensure that no duplicate values of the key column can exist? ?

a. KEY

b. UNIQUE

c. NODUPS

d. NODUPKEY

**3.** Which of the following will create a composite index for the table *Sasuser.Flightdelays?* ?
*(Sasuser.Flightdelays* contains the following columns: **`Date, FlightNumber, Origin, Destination, DelayCategory, DestinationType, DayOfWeek`**, and **`Delay`**.)

a. a.
```
proc sql;
 create index destination
     on sasuser.flightdelays(flightnumber,
                             destination);
```

b. b.
```
proc sql;
 create composite index places
     on sasuser.flightdelays (flightnumber,
                              destination);
```

c. c.
```
proc sql;
 create index on flightnumber,destination
    from sasuser.flightdelays (places);
```

d. d.
```
proc sql;
 create index places
     on sasuser.flightdelays (flightnumber,
                              destination);
```

**4.** Which of the following will write a message to the SAS log that shows whether PROC SQL has used an index? ?

a. a.
```
options msglevel=i;
proc sql;
 select *
    from sasuser.internationalflights
    where date between '01mar2000'd
          and '07mar2000'd;
```

b. b.
```
options index=yes;
proc sql;
 select *
    from sasuser.internationalflights
    where date between '01mar2000'd
          and '07mar2000'd;
```

c. c.
```
proc sql;
 select * (idxwhere=yes)
    from sasuser.internationalflights
    where date between '01mar2000'd
          and '07mar2000'd;
```

d. d.
```
proc sql;
 select * (msglevel=i)
    from sasuser.internationalflights
    where date between '01mar2000'd
          and '07mar2000'd;
```

**5.** Which of the following will drop (delete) an index from a table? ?

a. a.
```
proc sql;
  drop composite index flights
     from sasuser.marchflights;
```

b. b.
```
proc sql;
 delete index flights
    on sasuser.staffmaster(flightnumber, date);
```

c. c.
```
proc sql;
 drop index flights
    from sasuser.marchflights;
```

d. d.
```
proc sql;
 delete index
    on sasuser.marchflights(flightnumber,
                            flightdate);
```

**6.** Which of the following statements will show you all the indexes that are defined for a table? ?

a. DESCRIBE INDEX

b. DESCRIBE TABLE

c. SELECT

d. IDXNAME

**7.** What is the purpose of specifying the data set option IDXWHERE=YES? ?

a. It forces SAS to use the best available index to process the WHERE expression.

b. It creates an index from the expression in the WHERE clause.

c. It writes messages about index usage to the SAS log.

d. It stops SAS from using any index.

**8.** Which of the following is false regarding the use of an index? ?

a. Equijoins can be performed without internal sorts.

b. Indexes provide fast access to a small subset of data.

c. Indexes can be created for numeric columns only.

d. Indexes can enforce uniqueness.

**9.** Using an index is not likely to optimize a PROC SQL query in which of the following situations? ?

a. The query contains an IN subquery that references the key column.

b. The key column is specified in a WHERE clause expression that contains a comparison operator, the TRIM or SUBSTR function, the CONTAINS operator, or the LIKE operator.

c. The query is an equijoin, and all the columns in the join expression are indexed in one of the tables beingjoined.

d. The key column is specified only in a SELECT clause.

**10.** Which of the following is false regarding the IDXNAME= data set option? ?

a. The specified index must exist.

b. The specified index must be suitable by having at least its first or only column match a condition in the WHERE expression.

c. The option allows you to create and name an index on the table.

d. The option directs SAS to use an index that you specify.

### Answers

**1.** Correct answer: d

The index specified above is based on one column, so it is a simple index. In the CREATE INDEX statement, you specify the index name after the keywords CREATE INDEX. You do not include a keyword to specify that this is a simple index. The name of the key column is specified in parentheses after the table name. The name of a simple index must be the same as the name of the key column.

**2.** Correct answer: b

To create a unique index, the UNIQUE keyword is added to the CREATE INDEX statement, between the keywords CREATE and INDEX.

**3.** Correct answer: d

A composite index is based on two or more columns. In the CREATE INDEX statement, you specify the index name

after the keywords CREATE INDEX. You do not include a keyword to specify that this is a composite index. The names of the key columns are specified in parentheses after the table name. The name of a composite index cannot be the same as the name of any columns in the table.

**4.** Correct answer: a

Specifying the option MSGLEVEL=I causes informational messages about index usage to be written to the SAS log.

**5.** Correct answer: c

The DROP INDEX statement drops one or more specified indexes from a table. You specify the name of each index to be dropped after the keywords DROP INDEX. The table name is specified after the keyword FROM. The type of index and the names of the indexed columns are not specified in the statement.

**6.** Correct answer: b

The DESCRIBE TABLE statement lists all indexes for one or more tables that you specify, along with other information about the table(s).

**7.** Correct answer: a

The IDXWHERE=YES data set option tells SAS to use the best available index, even if the index does not optimize performance.

**8.** Correct answer: c

Indexes can be created on either character or numeric columns.

**9.** Correct answer: d

Using an index will optimize specific classes of PROC SQL queries. A query in which the key column is specified only in a SELECT clause is not one of these queries.

**10.** Correct answer: c

The IDXNAME= data set option directs PROC SQL to use an index that you specify. The specified index must exist and must be suitable by having at least its first or only column match the condition in the WHERE expression.